CLAIMS

1. (currently amended) A computer, comprising:

a binary translator programmed to translate at least a segment of a first binary representation of a program from a first binary representation in a first instruction set ~~architecture~~ to a second binary representation in a second instruction set ~~architecture~~, a sequence of side-effects in the second binary representation differing from a sequence of side-effects in the translated segment of the first binary representation, the second binary representation distinguishing individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory device(s);

instruction execution circuitry designed, while executing the second binary representation,

to identify an individual memory-reference instruction, or an individual memory reference of an instruction, a side-effect arising from the memory reference having been reordered by the translator, the memory reference having been believed at translation time to be directed to well-behaved memory but that at execution time is found to reference a device with a valid memory address that cannot be guaranteed to be well-behaved, based at least in part on an annotation encoded in a segment descriptor, and

based in the distinguishing, to identify whether the difference in sequence of side-effects may have a material effect on the execution of the program; and

circuitry and/or software designed to establish program state to a state equivalent to a state that would have occurred in the execution of the first binary representation, and to resume execution of the translated segment of the program in the first instruction set.

2. (currently amended) A method, comprising the step of:

for memory references generated as part of executing a stream of instructions on a computer, evaluating whether an individual memory reference of an instruction references a device having a valid memory address but that cannot be guaranteed to be well-behaved, based at least in part on an annotation encoded in a segment descriptor, a segment descriptor being data for controlling address formation by designating a segment base address, a segment length, and segment access control information.

3. (currently amended) The [[A]] method of claim 2, further comprising the step of:

if the memory reference cannot be guaranteed to be well-behaved, re-executing the instruction in an alternative execution mode.

4. (currently amended) The method of claim 3, further comprising the step of:

while translating at least a segment of a first binary representation of a program from a first instruction set ~~architecture~~ to a second instruction set ~~architecture~~ to produce the stream of instructions, annotating in the produced stream of instructions memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

5. (currently amended) The [[A]] method of claim 2, further comprising the step of:

for memory references generated as part of executing a stream of instructions on a computer, evaluating whether an individual memory reference of an instruction has been reordered relative to other side-effects in a manner that materially alters the execution of a program of the memory reference.

6. (currently amended) The method of claim 2, further comprising the step of:

if the memory reference cannot be guaranteed to be well-behaved, rolling back the state of the instruction stream to a prior state.

7. (previously presented) The method of claim 6, wherein:

the rolling back step is initiated when an exception occurs in an object program generated by binary translation from a reference implementation source program.

8. (currently amended) The [[A]] method of claim 7 [[6]], further comprising the step of:

resuming executing from the rolled back state, the resumed execution exccuting a precise side-effect emulation of the reference implementation.

9. (currently amended) The [[A]] method of claim 2, wherein the device having a valid memory address has an address in an I/O space of the computer.

10. (currently amended) The method of claim 2, further comprising the step of:

evaluating an annotation embedded in the instruction to determine whether the reference to the device that cannot be guaranteed to be well-behaved is to raise an exception.

11. (previously presented) The method of claim 2, further comprising the step of:

in circuitry embedded in address translation circuitry of the computer, evaluating whether the reference to the device that cannot be guaranteed to be well-behaved is to raise an exception.

12. (previously presented) The method of claim 2, wherein the segment descriptor is stored in a segment register.

13. (currently amended) The [[A]] method of claim 2, further comprising the step of:

forming the segment descriptor by copying another segment descriptor, and altering the annotation.

14. (currently amended) A computer, comprising:

instruction execution circuitry designed to evaluate, based at least in part on an annotation encoded in a segment descriptor, a segment descriptor being data for controlling address formation by designating a segment base address, a segment length, and segment access control information, whether an individual memory-reference instruction, or an individual memory reference of an instruction, references a device with a valid memory address that cannot be guaranteed to be well-behaved.

15. (currently amended) The [[A]] computer of claim 14, further comprising:

binary translator software programmed to generate the memory-reference instruction.

16. (currently amended) The [[A]] computer of claim 15, wherein the binary translator is further programmed to annotate the memory-reference instruction with an indication of whether the memory-reference instruction is likely or unlikely to reference well-behaved memory.

17. (currently amended) The computer of claim 14, further comprising:

a translator programmed to translate at least a segment of a source program into an object program, wherein a sequence of side-effects in the object program differs from a reference sequence of side-effects in the source program; and

circuitry and/or software designed to intervene during an execution of the object program on the computer to establish a program state equivalent to a state that would have occurred in the reference side-effect sequence, and to resume execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

18. (currently amended) The [[A]] computer of claim 14, further comprising ~~wherein~~:

code in a preamble of a program unit embracing the memory-reference instruction or the individual memory reference of an instruction that is designed to establish ~~establishes~~ a state of

the instruction execution circuitry, the instruction execution circuitry designed to raise an exception based on an evaluation of both the state and the evaluation of the reference to the device.

19. (currently amended) The [[A]] computer of claim 14, further comprising circuitry to raise an exception based on an evaluation of both an annotation embedded in the instruction and the evaluation of the reference to the device.

20. (currently amended) The [[A]] computer of claim 14, further comprising circuitry in an address translation path to raise an exception of a computer based on the evaluation of the reference to the device.

21. (currently amended) The [[A]] computer of claim 14, further comprising circuitry to raise an exception based on an evaluation of both a segment descriptor and the evaluation of the reference to the device.

22. (currently amended) A method, comprising the steps of:

while translating at least a segment of a first binary representation of a program from a first instruction set ~~architecture~~ to a second binary representation in a second instruction set ~~architecture~~, distinguishing individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory device(s);

while executing the second binary representation, identifying a load that was believed at translation time to be directed to well-behaved memory but that at execution time is found to be directed to non-well-behaved memory, based at least in part on an annotation encoded in a segment descriptor, and aborting the identified memory load, a segment descriptor being data for controlling address formation by designating a segment base address, a segment length, and segment access control information; and

based at least in part on the identifying, re-executing at least a portion of the translated segment of the program in the first instruction set.

23. (currently amended) The [[A]] method of claim 22, further comprising the steps of:

while executing the translation, detecting an ordering of side-effects that differs from the reference sequence of side-effects of the first binary representation in the first instruction set ~~architecture~~;

establishing the state of the translated program to a state equivalent to a state that would have occurred in the first binary representation in the first instruction set ~~architecture~~; and

resuming execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

24. (currently amended) The [[A]] method of claim 23, wherein the difference of ordering of side-effects includes a reordering of two side-effects relative to each other.

25. (currently amended) The [[A]] method of claim 23, wherein the difference of ordering of side-effects includes an elimination of a side-effect by the translating.

26. (currently amended) The method of claim 22, further comprising the step of:

annotating the second binary representation with an indication of the distinguishing drawn ~~distinction~~ between individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

27. (previously presented) The method of claim 22, further comprising the step of:

executing code in a preamble of the second binary representation to establish a state of instruction execution circuitry, the instruction execution circuitry designed to raise an exception based on an evaluation of both the state and the identification of loads.

28. (currently amended) The method of claim 22, further comprising the step of:

evaluating an annotation embedded in the instruction of the identified load to determine whether to raise an exception.

29. (previously presented) The method of claim 22, further comprising the step of:

in circuitry embedded in address translation circuitry, evaluating whether the instruction of the identified load is to raise an exception.

30. (currently amended) An apparatus, comprising:

a binary translator programmed to translate at least a segment of a first binary representation of a program from a first instruction set ~~architecture~~ to a second binary representation in a second instruction set ~~architecture~~, distinguishing individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory; and

instruction execution circuitry designed to execute the translated program in the second binary representation, and to identify, based at least in part on an annotation encoded in a segment descriptor, a segment descriptor being data for controlling address formation by designating a segment base address, a segment length, and segment access control information, memory loads that were believed at translation time to be directed to well-behaved memory but that at execution time are found to be directed to non-well-behaved memory, and to abort the identified memory load.

31. (currently amended) The [[A]] apparatus of claim 30, further comprising:

hardware designed to re-execute at least a portion of the translated segment of the program in the first instruction set.

32. (currently amended) The apparatus of claim 30, wherein:

the binary translator is further programmed to produce a sequence of side-effects in the second binary representation differing from a sequence of side-effects in the translated segment of the first binary representation; and

the instruction execution circuitry is further designed to identify cases during execution of the second binary representation in which the difference in sequence of side-effects may have a material effect on the execution of the program, to establish program state to a state equivalent to a state that would have occurred in the execution of the first binary representation, and to resume execution of the program from the established state in an execution mode that reflects the side-effect sequence of the first binary representation.

33. (previously presented) The apparatus of claim 32, wherein the difference of sequence of side-effects includes a reordering of two side-effects relative to each other.

34. (previously presented) The apparatus of claim 32, wherein the difference of sequence of side-effects includes an elimination of a side-effect by the translating.

35. (currently amended) The apparatus of claim 30, further comprising ~~the step of~~: software designed to annotate ~~annotating~~ the second binary representation with an indication of the distinguishing drawn ~~distinction~~ between individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

36. (currently amended) The [[A]] apparatus of claim 30, wherein a memory load that was believed at translation time to be directed to well-behaved memory, at execution time is found to be directed to non-well-behaved memory because it ~~device having a valid memory address~~ has an address in an I/O space of a computer.

37. (currently amended) The apparatus of claim 30, further comprising:

circuitry and/or software designed to evaluate ~~evaluating~~ an annotation embedded in the instruction of the identified load to determine whether to raise an exception.

38. (currently amended) The apparatus of claim 30, further comprising:

[[in]] circuitry embedded in address translation circuitry for the instruction execution circuitry designed to evaluate ~~, evaluating~~ whether the instruction of the identified load is to raise an exception.

39. (currently amended) The apparatus of claim 30, further comprising circuitry to raise an exception based on an evaluation of both a segment descriptor and the evaluation of the load directed to ~~reference~~ to non-well-behaved memory.

40. (currently amended) A method, comprising the steps of:

translating at least a segment of a source program into an object program, the source program instructing a reference execution with a reference sequence of side-effects, the object program instructing an execution in which the sequence of side-effects differs from the reference side-effect sequence;

during an execution of the object program on a computer, detecting a side-effect about to be committed to processor state in which the differing side-effect sequence may have a material effect on the execution of the program, and aborting the side-effect;

establishing a program state equivalent to a state that would have occurred in the reference execution; and

resuming execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

41. (currently amended) A method of claim 40, wherein the source program is coded in a first instruction set ~~architecture~~, and the object program is coded in a second instruction set ~~architecture~~.

42. (currently amended) The method of claim 41, further comprising the steps of:

evaluating, based at least in part on an annotation encoded in a segment descriptor, whether an individual memory reference of an instruction initiated by execution of the object program references a device having a valid memory address but that cannot be guaranteed to be well-behaved; and

initiating the establishing step based at least in part on the evaluating.

43. (previously presented) The method of claim 41, further comprising the step of:

annotating the object program with an indication of a distinction between individual memory references that are believed to be directed to well-behaved memory from memory references that are believed to be directed to non-well-behaved memory.

44. (previously presented) The method of claim 41, further comprising the step of:

executing code in a preamble of the object program to establish a state of instruction execution circuitry, the instruction execution circuitry designed to raise an exception based on an evaluation of both the state and the evaluation of individual memory references.

45. (currently amended) The method of claim 41, further comprising the step of:

evaluating an annotation embedded in the instruction of the ~~individual~~ side-effect about to be committed to determine whether to raise an exception.

46. (currently amended) The method of claim 41, further comprising the step of:

in circuitry embedded in address translation circuitry of the computer, evaluating whether the instruction of the ~~individual~~ side-effect about to be committed is to raise an exception.

47. (currently amended) The [[A]] method of claim 41, further comprising the step of:

raising an exception based on an evaluation of both a segment descriptor and the evaluation of the side-effect.

48. (currently amended) The [[A]] method of claim 41, further comprising the step of:

evaluating an annotation encoded in a segment descriptor to determine whether the side-effect about to be committed ~~reference to the non-well behaved device~~ is to raise an exception.

49. (currently amended) The [[A]] method of claim 48 [[41]], further comprising the step of:

forming the segment descriptor by copying another segment descriptor, and altering the annotation.

50. (currently amended) The [[A]] method of claim 49 [[41]], further comprising the step of:

copying into the formed segment descriptor a variable indicating an assumed sensitivity of the translation to alteration of the sequence of side-effects.

51. (currently amended) The [[A]] method of claim 41, wherein the difference ~~of ordering~~ of side-effects includes a reordering of two side-effects relative to each other.

52. (currently amended) The [[A]] method of claim 41, wherein:

the establishing step is initiated when an exception occurs in the object program.

53. (currently amended) The method of claim 41, further comprising the step of:

resuming executing from the established state, the resumed execution executing a precise side-effect emulation of the reference side-effect sequence ~~execution~~.

54. (currently amended) The [[A]] method of claim 41, further comprising the step of:

using a descriptor generated during the translation to establish a pre-exception reference state.

55. (currently amended) An apparatus, comprising:

a binary translator programmed to translate at least a segment of a program from a first binary representation in a first instruction set ~~architecture~~ to a second binary representation in a second instruction set ~~architecture~~, a sequence of side-effects in the second binary representation differing from a sequence of side-effects in the translated segment of the first binary representation; and

instruction execution circuitry and/or software designed to

identify cases during execution of the second binary representation in which the difference in sequence of side-effects may have a material effect on the execution of the program, before committing the side-effect to processor state, and aborting the side-effect; and

to establish a program state equivalent to a state that would have occurred in the execution of the first binary representation, and to resume execution of the program from the established state in an execution mode that reflects the side-effect sequence of the first binary representation.

56. (currently amended) The [[A]] apparatus of claim 55, wherein ~~further comprising~~:

the difference in side-effect sequence annotated by the binary translator includes an annotation of [[annotating]] the second binary representation with an indication of a [[the]] distinction between individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

57. (currently amended) The apparatus of claim 56, wherein:

the instruction execution circuitry is designed to evaluate an annotation embedded in the instruction of the identified case of a side-effect ~~load~~ to determine whether to raise an exception.

58. (currently amended) The apparatus of claim 56, further comprising:

[[in]] circuitry embedded in address translation circuitry for the instruction execution circuitry designed to evaluate ~~, evaluating~~ whether the instruction of the identified case of a side-effect ~~load~~ is to raise an exception.

59. (previously presented) The apparatus of claim 56, further comprising:

circuitry to evaluate an annotation encoded in a segment descriptor to determine whether a reference to a device in non-well-behaved memory is to raise an exception.

60. (cancelled)

61. (previously presented) The apparatus of claim 55, wherein the difference of sequence of side-effects includes a reordering of two side-effects relative to each other.

62. (previously presented) The apparatus of claim 55, wherein the difference of sequence of side-effects includes an elimination of a side-effect .

63. (previously presented) The apparatus of claim 55, wherein the difference of sequence of side-effects results from combining two side-effects in the binary translator.

64. (previously presented) The apparatus of claim 55, wherein:

the establishing is initiated when an exception occurs in the second binary representation.

65. (previously presented) The apparatus of claim 55, wherein:

the resumed execution executes a precise side-effect emulation of the first binary representation.